



VistAWeb

Technical Manual

*Version 5.0 (Patch WEBV*1*7)*

July 2006

Department of Veterans Affairs
VistA Health Systems Design & Development,
Health Data Systems, Current Systems

Revision History

Date	Patch	Page(s)	Change(s)	Project Manager	Technical Writer
7/11/06	WEBV*1*7	2, 4, 5, 15 App A App B	Removed references to VistAWebDocs application. Added two new scripts to Appendix A. Removed appendix B, which makes the former appendix C now B.	S. Madsen	M. Kelsey
3/20/06	WEBV*1*5	15, App B	The <i>Using VistAWeb User Management Web Application</i> section was modified to reflect a new method of requesting Special User access. Some new file names (CAPRI files, for example) have been added to Appendix B; <i>testSecurityKey.java</i> filename was deleted.	S. Madsen	M. Kelsey
8/22/05	WEBV*1*4	1, 7, 8, App C	Remote data from HDR and CHDR, change in login process flow patient synchronization with CPRS, new htm, aspx, aspx.cs, and aspx,resx sample code added.	G. Smith	M. Kelsey
5/05/05	n/a	All	Replaced URLs and made format changes.	G. Smith	M. Kelsey
2/25/05	n/a	All	Reviewed for release		
2/18/05	n/a	2, 6, & 28	Removed URL references (VISN CIO will provide)		
2/10/05	n/a	14, 16, 21, & 26	Removed reference to Special User Script		
1/31/05	Informational Patch number OR*3*230	All	Initial User Manual for use with beta test version		

Table of Contents

REVISION HISTORY	II
TABLE OF CONTENTS	III
INTRODUCTION	1
Figure 1: VistAWeb Application Tier Interaction	2
ASSUMPTIONS.....	2
SYSTEM REQUIREMENTS	3
HARDWARE.....	3
Components that Apply to Both Web and Database Servers.....	3
Web Server Components	3
Database Server Components	3
SOFTWARE	3
Figure 2: Web Service Extension Settings	4
Figure 3: VistAWeb Website and Accompanying Web Application	5
VISTAWEB OVERVIEW.....	5
LANGUAGE SPECIFICATIONS.....	6
USING VISTAWEB	6
Standalone Application Process	6
CPRS-Spawned VistAWeb Process	7
Figure 4: VistAWeb Login Process Flow	8
VISTAWEB UNDER THE HOOD.....	9
ASP.NET / Code-Behind Example: Insurance.aspx	9
Figure 5: VistAWeb Project References	11
RELEASING PROJECT UPDATES TO PRODUCTION	14
OTHER VISTAWEB MANNERISMS.....	15
USING VISTAWEB USER MANAGEMENT WEB APPLICATION	15
Process.....	15
SQL SERVER DATABASE OVERVIEW	16
APPENDIX A: DATABASE SCHEMA	17
LOG CREATION SCRIPT	17
CPRSUSERS CREATION SCRIPT.....	17
SPECIALUSERS CREATION SCRIPT	18
REQUESTS CREATION SCRIPT	18
LOGDESC VIEW CREATION SCRIPT	18
APPENDIX B: VISTAWEB CODE SAMPLES.....	19
SAMPLE HTM (COUNTDOWN.HTM):.....	19
SAMPLE ASPX PAGE (TEXTRECORD.ASPX):.....	20
SAMPLE CODE-BEHIND PAGE (TEXTRECORD.ASPX.CS):.....	22
SAMPLE TEMPLATE PAGE (TEXTRECORD.ASPX.RESX).....	26

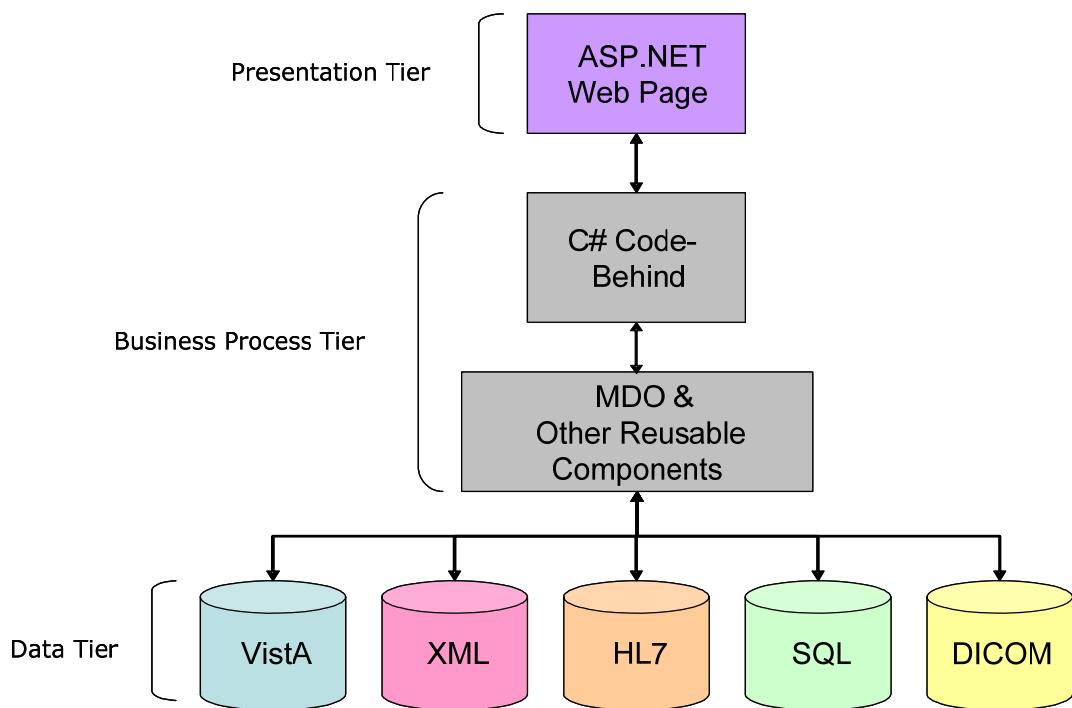
Introduction

Veterans Health Information Systems and Technology Architecture (VistA) VistAWeb is an intranet web application used to review remote patient information found in VistA, as well as in Health Data Repository (HDR) databases and the Department of Defense (DoD) Clinical Data Repository/Health Data Repository (CHDR) database. VistAWeb mirrors the behavior of the Computerized Patient Record System (CPRS) and Remote Data View (RDV). However, by permitting a more robust and timely retrieval of remote-site patient data, VistAWeb is also an enhancement to CPRS/RDV.

There are two ways to access VistAWeb. VistAWeb can be made available from the Tools Menu within CPRS; or, as a secondary standalone option, it can be accessed by entering the uniform resource locator (URL) link (<https://vistaweb.med.va.gov/>) in the Internet Explorer address bar. Unlike the standalone version of VistAWeb, the CPRS-spawned version is compliant with the Health Level 7 (HL7) Clinical Context Object Workgroup (CCOW) standards and therefore maintains context with the patient selected in CPRS.

The design of VistAWeb uses n-tier architectural principles, where VistAWeb represents the presentation tier (ASP.Net Web Page). The business process tier is represented by multiple components that VistAWeb uses to access the data tier. Business process components include such elements as code-behind pages (a programming feature of Microsoft.NET programming), Medical Domain Objects (MDO), and a collection of other reusable components (.dll files) such as MDO will be discussed in other technical documents. The data tier comprises multiple data sources, such as VistA, XML, and numerous SQL-compliant relational databases (e.g., Oracle, Microsoft SQL Server, and Microsoft Access). Although some code-behind pages do interact with an SQL-compliant database and some XML sources, the rest of the data tier interactions take place in reusable components such as MDO. [Figure 1](#) depicts, at a high level, the interaction of the different application tiers.

Figure 1: VistAWeb Application Tier Interaction



Assumptions

The intended audience of this Technical Guide is system administrators, specifically web administrators. Readers are assumed to possess the technical knowledge of programming principles using languages such as Java, XML, C#, HTML and SQL. Additionally, users of this manual should also possess the technical knowledge of how to configure and interact with application servers. This document also assumes the necessary security hardening guidelines have already been implemented. See the Office of Cyber and Information Security link below for information pertaining to security requirements:

https://vaww.ocis.va.gov/portal/server.pt?in_hi_opt_comm_community=257&in_hi_space=SearchResult&in_hi_control=bannerstart&in_hi_userid=2&in_se_sel_1=everything&in_tx_query=hardening+guidelines

For additional information on technologies and principles used in the development and implementation of VistAWeb, the following sources are recommended reading:

- ASP.NET: <http://www.asp.net/Default.aspx?tabindex=0&tabid=1>
- C#: <http://msdn.microsoft.com/vcsharp/>
- Java: <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html>
- Java Design Patterns: <http://java.sun.com/developer/technicalArticles/J2EE/patterns/>
- .NET Application Development / n-tier: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdadotnetarch001.asp>
- SQL: <http://www.w3schools.com/sql/default.asp>
- World Wide Web Consortium: <http://www.w3.org/>

The remainder of this document is divided into the following sections:

- System Requirements
- VistAWeb Overview
- SQL Server Database Overview
- Appendixes

System Requirements

Hardware

The servers that run VistAWeb are configured in Silver Spring, Maryland. The exact details of all components are unknown, but the basic components for the web servers and the database servers are listed below.

Components that Apply to Both Web and Database Servers

- Dell PowerEdge 4210 Rack with KVM (16-port switch)
- Two SAN adapter cards and Emulex FC HBA with Power Path licenses

Web Server Components

- Two Dell PowerEdge 6650s
- Dual-CPU 2.2 GHz processor
- 8 GB RAM
- Five 36-GB SCSI hard drives
- Dual network interface cards

Database Server Components

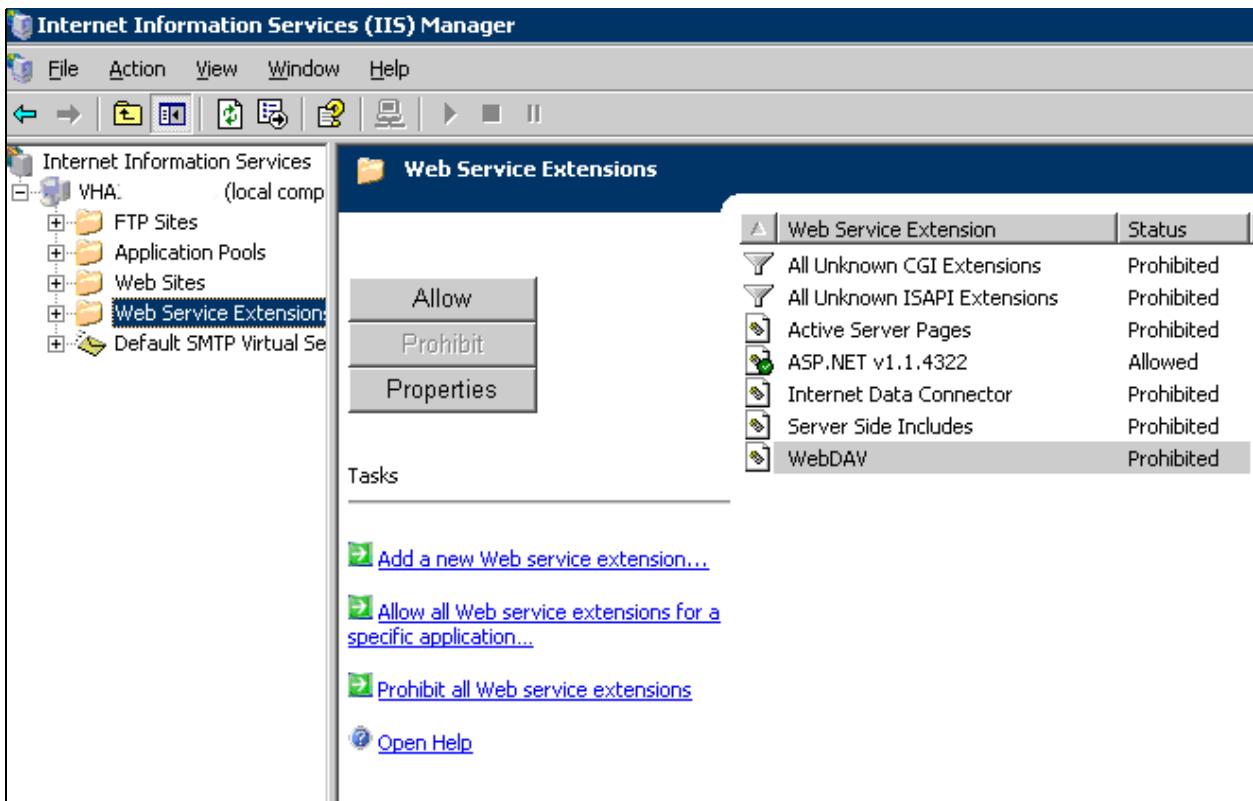
- Two Dell PowerEdge 6650s
- Quad-CPU 2.2 GHz processors
- 8 GB RAM
- Five 36-GB SCSI hard drives
- Dual network interface cards

Software

- Windows Server 2003 Enterprise configured with the role of Application Server
- Internet Information Services (IIS) 6.0 (installed by default as part of the Application Server role)
- Microsoft Visual J#.NET 2003 runtime component
- .NET Framework 1.1 (part of the Windows Server 2003 operating system default installation)
- FTP services and an FTP folder (to be used as a staging location for updates to VistAWeb)
- SMTP Virtual Server
- .NET Framework 1.1 is installed by default on Windows 2003 systems. Services packs and updates to all three components are available through Microsoft Windows update (<http://windowsupdate.microsoft.com>).
- Web Extension Services set to allow ASP.NET extensions (see [Figure 2](#))

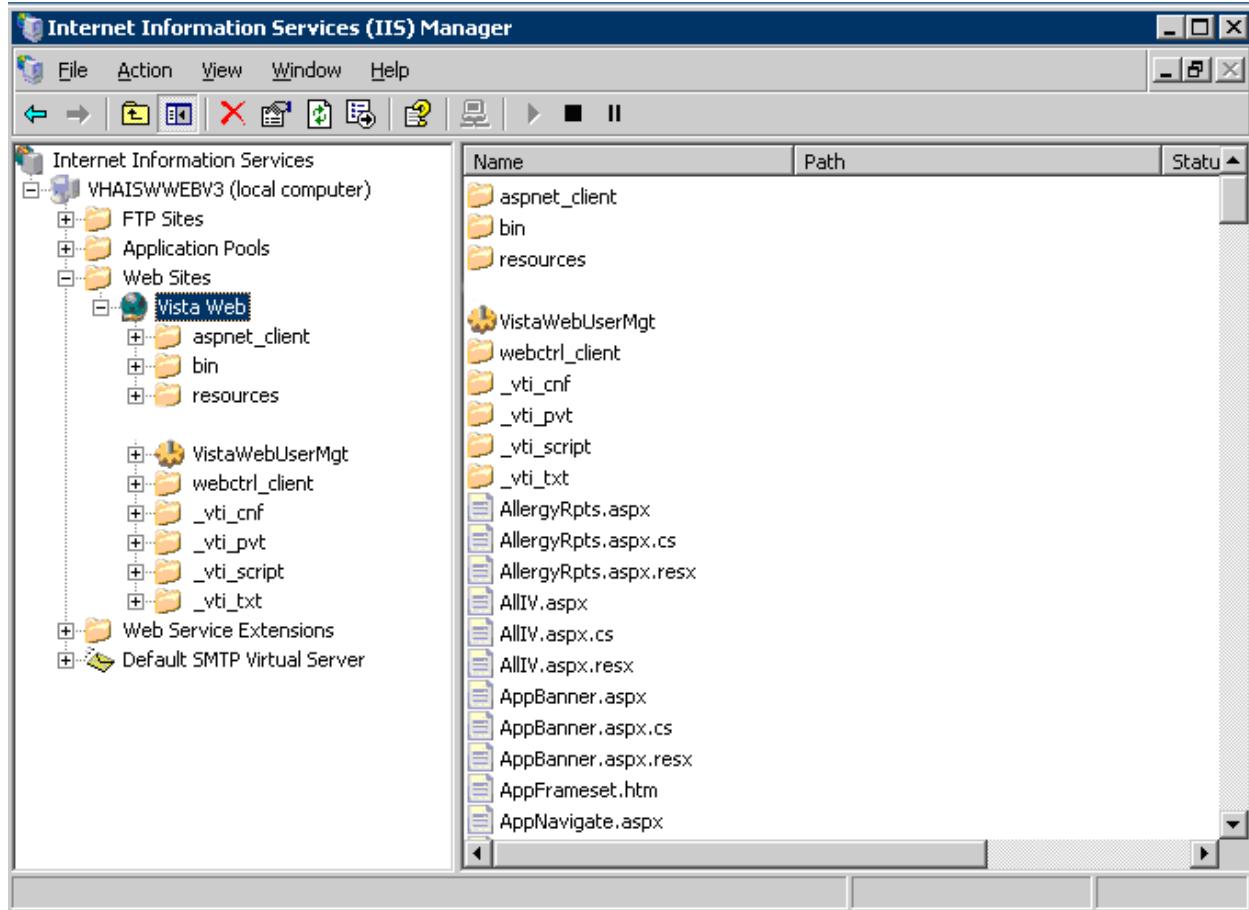
- SQL Server 2000 (The database does not need to run on the same server as the web application.)

Figure 2: Web Service Extension Settings



Note that there is one other application that will run alongside VistAWeb: VistAWebUserMgt, depicted in [Figure 3](#). VistAWebUserMgt is a web application that allows select users to control VistAWeb user access to patients at VA sites other than a user's local site.

Figure 3: VistAWeb Website and Accompanying Web Application



For instructions on installing and configuring VistAWeb and the related intranet applications, see the VistAWeb Installation Guide.

VistAWeb Overview

VistAWeb's web pages are a mix of basic HTML pages and .NET Active Server Pages (ASP.NET), noted by the respective extensions of .htm (or .html) and .aspx. HTM and HTML pages are standalone pages. However, for the ASP.NET pages, there are actually three pages to be aware of:

1. aspx pages are written in HTML and ASP.NET tags;
2. aspx.cs pages represent the “code-behind” pages that power the aspx pages; and
3. aspx.resx pages are template files automatically created for each aspx page.

Note: The aspx.resx pages are presently not used beyond their default .NET internal settings, and, therefore, will not be documented at this time. A code sample is provided in Appendix B, which contains a sample code from each of the four—htm, aspx, aspx.cs, aspx.resx.

Language Specifications

The languages used in VistAWeb (and subsequently VistAWebUserMgt) are:

- ASP.NET
- HTML, including Cascading Style Sheets (CSS)
- XML
- C#
- JavaScript

Using VistAWeb

VistAWeb is an intranet application that may be accessed as a standalone application, or spawned from the CPRS tools menu. VistAWeb's patient data menu is very similar to the report menu found in the CPRS Reports tab. Users may wish to use either the standalone application or the CPRS-spawned version from the CPRS Tools menu. VistAWeb users must have an active existing CPRS account with the necessary CPRS contexts enabled. The two methods for using VistAWeb are documented below. For more comprehensive documentation on the use of VistAWeb, please read the VistAWeb User Manual.

Standalone Application Process

- User launches the web browser application (e.g., Internet Explorer).
- User enters the URL of VistAWeb (<https://vistaweb.med.va.gov>) in the address bar and presses the Enter key or clicks the mouse cursor on the “Go” button adjacent to the address bar.
- VistAWeb loads into the user’s web browser.
- User must select a login site link on the left of the display screen by clicking the mouse cursor on the desired site where the user has access.
- User is provided with the VistA Login screen.
- User enters his or her CPRS access/verify codes in the spaces provided and presses the Enter key or clicks the mouse cursor on the Login button.
- Once the user’s account is authenticated against CPRS, the user’s remote site patient selection permissions are verified from a SQL Server database. The permissible sites for the user to choose patients from are then displayed.
- User must select a site from which to select a patient if selecting a site other than his or her default site.
- User is presented a Patient Selection screen for entering the desired patient name. User then clicking the mouse cursor on the “Find” button.
- The Sites and Notices screen is displayed, which identifies the sites where the patient has been seen.
- User can look at different elements of the patient record by selecting a desired report from the list of available reports on the left side of the displayed screen.

By default, a VistAWeb user is permitted to select patients that are in the local VistA system where the user logs in. VistAWeb will retrieve data for these patients from all sites where the patients have visited. Some users (researchers or referral coordinators, for example) may need to select patients that are not in the local VistA. These users must be granted Special User access. Special User access can be granted for one site in addition to the login site,

several sites, an entire VISN, or all sites nationally. The process for requesting special user permission is documented in the VistAWeb User Manual.

Note that regardless of which site is selected for a patient (local or remote), once a patient has been selected, VistAWeb uses the Master Patient Index (MPI) at the selected site to determine at what other sites the patient has remote data and establishes the connections to each of those sites to retrieve data requested by the user. Also note that when the connections are established, if there are any remote sites that have data, the remote site is checked to see if the VistAWeb user has ever visited the site before. If the user has not, then a visitor account is created for the user at the remote site. The full details of how the visitor account creation occurs will be fully explained in future documentation of MDO. What this means to VistAWeb users, however, is that initially VistAWeb may appear to run a little slow, but over time it will begin to increase in speed. This is because VistAWeb is creating visitor accounts for the user at the remote site; in future visits by the user to the remote site, such account creations are not necessary.

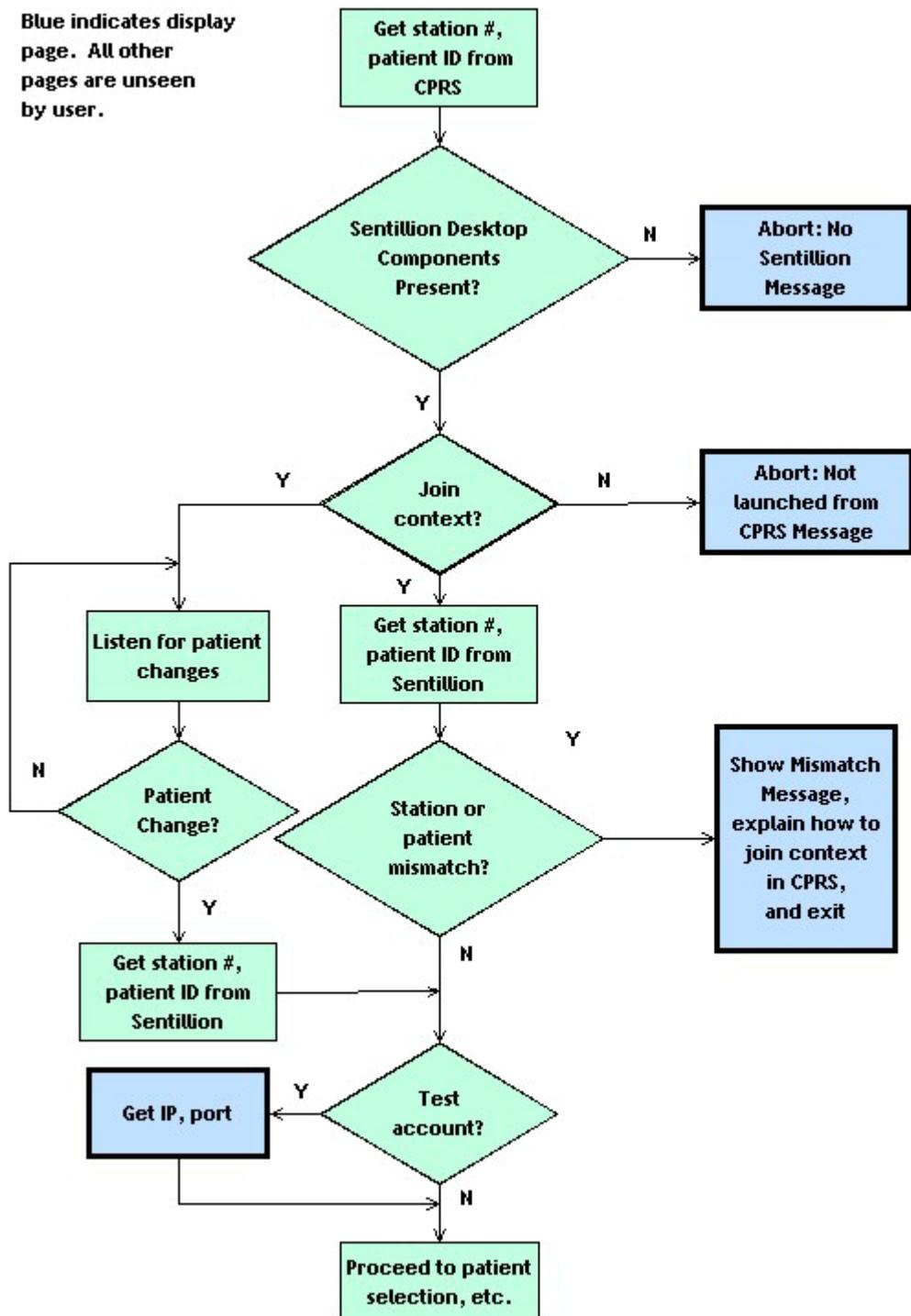
CPRS-Spawned VistAWeb Process

- User opens CPRS.
- User enters his or her CPRS access/verify code.
- User selects a patient.
- User launches VistAWeb from the Tools menu
- VistAWeb opens in a web browser window (e.g., Internet Explorer) and one of two things occurs:
 1. If the proper Sentillion Vergence controls are installed, VistAWeb will synchronize and display the CPRS-selected patient if the session of CPRS that launched VistAWeb is in context. The user can now look at different elements of the patient record by selecting the desired report from the reports menu on the left side of the user's display screen.
 2. If the components are not installed, or if the CPRS session is not in context, then the user is warned that either the components are not installed, or that CPRS lacks proper context synchronization, and the user is forced to close VistAWeb. After either or both of these situations are rectified, the user can start the process anew and display the patient records using VistAWeb, as in number 1 above.

Note: Unlike the standalone version of VistAWeb, the launched version from the CPRS Tools menu does not permit the user to change patients from within VistAWeb. VistAWeb is CCOW compliant and, therefore, maintains context with the patient who was selected in CPRS. Users must have the Sentillion Vergence Locator application loaded on the desktop and must maintain patient synchronization (context) to use the CPRS-spawned VistAWeb process.

[Figure 4](#) shows the process flow for the login and use scenarios discussed above.

Figure 4: VistAWeb Login Process Flow



VistAWeb under the Hood

Like all applications developed using Active Server Pages (ASP), all data source communications and most complex tasks are performed on the server side rather than the client side. The client (i.e., the VistAWeb user) only sees the end result of what the server does. What the client sees is standard HTML and client-side scripting generated by the server. So even though a common ASP.NET tag of <asp:textbox> might exist in an aspx page on the server, what is shown in the client's browser is <input type=text>. More importantly, the client never sees or knows how the data is collected or from where it is collected.

VistAWeb communicates with VistA and other data sources using a collection of methods. All data source communication methods are done through code-behind pages and through reusable components. Both the code-behind pages (.cs files) and the reusable components (.dll files) represent the business process tier. To provide better context and clarity, let's examine a code-behind page and a web page in greater detail.

ASP.NET / Code-Behind Example: Insurance.aspx

Insurance.aspx is a basic ASP.NET page, but the coding techniques are the same for the more complex ASP.NET pages in VistAWeb. Note that only the unique elements of ASP.NET will be explained here.

```
<%@ Page language="c#" Codebehind="Insurance.aspx.cs" AutoEventWireup="false"
Inherits="EMR.Insurance" %>
```

Every .aspx page must identify which code-behind page it must link to. In this case, it is Insurance.aspx.cs. It is possible to use C# programming in-line the same way that VBScript and JavaScript can be used in regular ASP (i.e., pre-.NET) pages, but VistAWeb only uses C# in code-behind pages.

```
<asp:Panel CssClass="divGrid" ID="plPanel" Runat="server">
    <asp:DataGrid id="DataGrid1" runat="server" Font-Names="Georgia" Font-
    Size="8pt" AutoGenerateColumns="False"
        AllowSorting="True" UseAccessibleHeader="True">
            <AlternatingItemStyle Font-Size="8pt" Font-Names="Georgia"-
            BackColor="Ivory"></AlternatingItemStyle>
            <HeaderStyle Font-Names="Trebuchet MS" Font-Bold="True"-
            ForeColor="White" BackColor="DarkSlateGray"></HeaderStyle>
            <Columns>
                <asp:BoundColumn DataField="Company"
                    SortExpression="Company,VistaTS" ReadOnly="True"-
                    HeaderText="Company"></asp:BoundColumn>
                    <asp:BoundColumn DataField="Policy"
                        SortExpression="Policy,Company,VistaTS" ReadOnly="True"-
                        HeaderText="Policy Type"></asp:BoundColumn>
                            <asp:BoundColumn DataField="GroupNum" ReadOnly="True"-
                                HeaderText="Group #"></asp:BoundColumn>
                                <asp:BoundColumn DataField="Holder" ReadOnly="True"-
                                    HeaderText="Holder"></asp:BoundColumn>
                                    <asp:BoundColumn DataField="EffectiveDate"
                                        SortExpression="VistaTS" ReadOnly="True"-
                                        HeaderText="Effective Date"></asp:BoundColumn>
                                            <asp:BoundColumn DataField="ExpirationDate"-
                                                ReadOnly="True" HeaderText="Expiration Date"></asp:BoundColumn>
```

```
<asp:BoundColumn DataField="Facility"
SortExpression="Facility,VistaTS,Company" ReadOnly="True"
HeaderText="Site"></asp:BoundColumn>
<asp:BoundColumn Visible="False" DataField="VistaTS"
ReadOnly="True" HeaderText="TS"></asp:BoundColumn>
</Columns>
</asp:DataGrid>
</asp:Panel>
```

The ASP.NET web controls, when read by the server, are rendered as HTML tags when transmitted to the client. For example, the `<asp:Panel>` tag will be rendered as a `<div>` tag. The `<asp:DataGrid>` tag will be rendered as a table. Note that ASP.NET web controls can be read by the C# code-behind pages before the page is rendered, while the regular HTML tags cannot. This is because all ASP.NET and C# code is interpreted first, and HTML controls are interpreted last. In fact, HTML controls cannot be read by the C# code, because the controls do not exist prior to the page being loaded. The code-behind is executed prior to the .aspx page.

To explain more fully, let's examine the code-behind for Insurance.aspx.

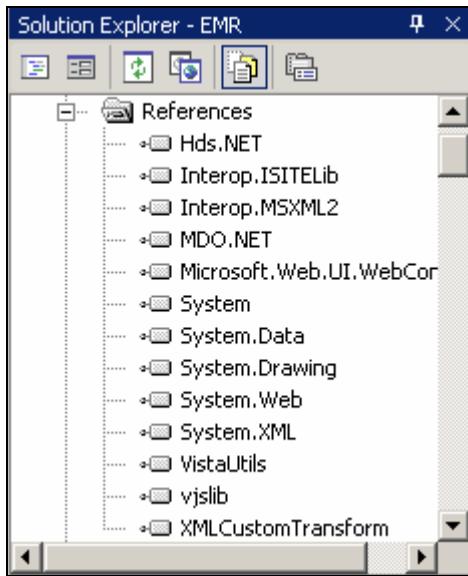
```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using gov.va.med.vistaweb.util;
```

At the beginning of every code-behind page, there are usually several lines of code that identify which additional class libraries should be included and referenced. Including these libraries permits the programmer to interact with the routines within the libraries. For example, the `System.Web.UI.WebControls` Library allows the programmer to interact with the methods and properties of the ASP.NET controls and receive programming assistance from the IDE syntax checker. In order to interact with specific ASP.NET web controls, the controls must be identified as shown below:

```
protected System.Web.UI.WebControls.DataGrid DataGrid1;
protected System.Web.UI.WebControls.Literal TextArea;
protected System.Web.UI.WebControls.Panel plPanel;
```

Note that MDO is not included in the “using” list above. This is because it is referenced at the project level, as shown in [Figure 5](#).

Figure 5: VistAWeb Project References



A major difference between ASP and ASP.NET pages is that ASP.NET pages are self-submitting, meaning that they do not submit their data to other ASP.NET pages. In the pre-.NET days, ASP pages usually submitted their data to a different ASP page, with the different page being identified in the HTML tag `<form onaction="secondasppage.asp" method="post">`. In ASP.NET, the `onaction` event of the `form` tag is ignored, and clicking a submit button will submit the form's data to itself. With this paradigm shift, additional programming elements were needed for the code-behinds to prevent certain pitfalls. For example, say the `Insurance.aspx` page is being visited for the first time by the user. Certain values are queried and displayed on the page. Without a conditional element being included in the `Page_Load` event, any reload would continually reload the original values, thereby wiping out any alterations to the page made by the user (assuming any were allowed). This conditional element sample is identified below:

```
private String sessionPageName = "";
private String sessionDataTableName = "";
private String sessionDataViewName = "";
private void Page_Load(object sender, System.EventArgs e)
{
    Response.CacheControl = "no-cache";
    Response.AddHeader("Pragma", "no-cache");
    Response.Expires = -1;
    sessionPageName = sender.GetType().FullName;
    sessionDataTableName = sessionPageName + "DataTable";
    sessionDataViewName = sessionPageName + "DataView";
    if (!Page.IsPostBack)
    {
        DataTable dataTable = new DataTable();
        Session[sessionDataTableName] = dataTable;
        defineGrid();
        query();
    }
}
```

Having the !Page.IsPostBack condition ensures that the query is performed only once rather than each time the page is reposted to itself.

Let's continue examining this code. The first three "Response" lines of the Page_Load event prevent the page from being cached in the client's Temporary Internet Files folder. During the initial load of the Insurance page, a DataTable object is created. It is assigned to a Session object. Next, the defineGrid() event is executed:

```
private void defineGrid()
{
    DataTable dataTable = (DataTable)Session[sessionDataTableName];
    dataTable.Columns.Add(new
    DataColumn("Company", Type.GetType("System.String")));
    dataTable.Columns.Add(new
    DataColumn("Policy", Type.GetType("System.String")));
    dataTable.Columns.Add(new
    DataColumn("GroupNum", Type.GetType("System.String")));
    dataTable.Columns.Add(new
    DataColumn("Holder", Type.GetType("System.String")));
    dataTable.Columns.Add(new
    DataColumn("EffectiveDate", Type.GetType("System.String")));
    dataTable.Columns.Add(new
    DataColumn("ExpirationDate", Type.GetType("System.String")));
    dataTable.Columns.Add(new
    DataColumn("Facility", Type.GetType("System.String")));
    dataTable.Columns.Add(new
    DataColumn("VistaTS", Type.GetType("System.String")));
}
```

This function builds the data table object's columns. Note that the columns are identical to the <asp:DataGrid> columns documented earlier.

Next, the query() event is executed:

```
private void query()
{
    try
    {
        MDO.InsuranceRpt[] rex = getRex();
        populateGrid(rex);
    }
    catch (Exception ex)
    {

        Global.showOops((MDO.User)Session["User"],(MDO.Patient)Session["Patient"],
                        this.Context.Request,ex,this.Context.Response);
    }
    Log.write(Request,Session,null,"From " +
    ((MDO.MultiSiteDAO)Session["MultiSiteDAO"]).getSiteList());
}
```

Note here the insurance object being created. The retrieval of the patient's insurance data is being handled by the one line:

```
MDO.InsuranceRpt[] rex = getRex();
```

The MDO object MDO.InsuranceRpt[] rex is populated by another C# function getRex(), shown below:

```

private MDO.InsuranceRpt[] getRex()
{
    MDO.Patient patient = (MDO.Patient)Session["Patient"];
    try
    {
        MDO.InsuranceRpt[] rex = patient.getInsuranceRpts();
        if (rex == null)
        {

            ((MDO.MultiSiteDAO)Session["MultiSiteDAO"]).setInsuranceRpts(patient);
        }
    }
    catch (Exception ex)
    {

        Global.showOops((MDO.User)Session["User"],(MDO.Patient)Session["Patient"],
                        this.Context.Request,ex,this.Context.Response);
    }
    return patient.getInsuranceRpts();
}

```

First, the function retrieves the patient object from the Session object Session[“Patient”]. The syntax “(MDO.Patient)” ensures that the Session[“Patient”] object is cast properly. Next, the function checks to see if the insurance data has already been retrieved from the different data sources. If not, the line of syntax containing “setInsuranceRpts(patient)” collects the insurance data from the different data sources. Any errors that occur during the data retrieval will send an error message to the VistAWeb Technical Support Group. Otherwise, the getRex() function returns the patient’s Insurance reports to the calling statement in the query() function.

Upon returning to the query() function,

```

        populateGrid(rex);
    }
    catch (Exception ex)
    {

        Global.showOops((MDO.User)Session["User"],(MDO.Patient)Session["Patient"],
                        this.Context.Request,ex,this.Context.Response);
    }
    Log.write(Request,Session,null,"From " +
    ((MDO.MultiSiteDAO)Session["MultiSiteDAO"]).getSiteList());
}

```

The data table is filled and the data grid is populated using the function populateGrid(rex):

```

private void populateGrid(MDO.InsuranceRpt[] rex)
{
    if (rex.Length == 0)
    {
        DataGrid1.Visible = false;
        plPanel.Visible = false;
        TextArea.Text = "No insurance reports were found";
        return;
    }
    else
    {

```

```

        DataGrid1.Visible = true;
        p1Panel.Visible = true;
    }
    MDO.SiteTable siteTable = (MDO.SiteTable)Session["VhaSiteTable"];
    DataTable dataTable = (DataTable)Session[sessionDataTableName];
    dataTable.Clear();
    TextArea.Text = "";
    for (int i=0; i<rex.Length; i++)
    {
        DataRow dataRow = dataTable.NewRow();
        dataRow[COMPANY_COL] = rex[i].company;
        dataRow[POLICY_COL] = rex[i].policy;
        dataRow[GROUPNUM_COL] = rex[i].groupNum;
        dataRow[HOLDER_COL] = rex[i].holder;
        dataRow[EFFECTIVE_DATE_COL] = rex[i].effectiveDate;
        dataRow[EXPIRATION_DATE_COL] = rex[i].expirationDate;
        String[] parts = StringUtils.Split(rex[i].facility, StringUtils.SEMICOLON);
        dataRow[FACILITY_COL] = parts[0];
        dataRow[VISTA_TS_COL] =
            StringUtils.DateTimeString2VistaTimestamp(rex[i].effectiveDate);
        dataTable.Rows.Add(dataRow);
    }
    DataView dataView = new DataView(dataTable);
    dataView.Sort = "VistaTS Desc";
    DataGrid1.DataSource = dataView;
    DataGrid1.DataBind();
    Session[sessionDataViewName] = dataView;
}

```

If there is no data to show, then the data grid and the <asp:panel> objects are set to not be visible. Making this setting to the objects means that the objects will not be rendered as HTML and therefore never be known to exist by the client. It is important to understand the distinction between the visible attribute and the hidden attribute. The hidden attribute of HTML controls means that the control is rendered on the client side, while the visible = false attribute of asp.net controls means that no aspect of the control even shows up on the client side.

Most of the VistAWeb data grids support a sorting feature, meaning the user can click on the header of a column and sort the data grid by the selected column.

```

private void DataGrid1_SortCommand(object source,
System.Web.UI.WebControls.DataGridSortCommandEvent e)
{
    TextArea.Text = "";
    DataView dataView = (DataView)Session[sessionDataViewName];
    dataView.Sort = e.SortExpression;
    DataGrid1.DataSource = dataView;
    DataGrid1.DataBind();
}

```

Releasing Project Updates to Production

Whenever there is a change to the VistAWeb EMR project, the project must be recompiled. This project recompilation produces a .dll file. This .dll file, unlike the pre-.NET days of ASP and middle-tier development, does not require a component object model (COM) wrapper,

but is instead COM-less. Bottom line—this means faster application development, faster deployment, and less application server downtime. Unlike the pre-.NET days when the IIS services would have to be stopped to release .dll updates, in .NET, the IIS services do not need to be stopped, because the COM-less .dll is not locked and can be easily overwritten.

Other VistAWeb Mannerisms

- CCOW—VistAWeb is Clinical Context Object Workgroup (CCOW) compliant and therefore maintains context with the patient who was selected in CPRS. VistAWeb retrieves data for that patient from all sites where the patient has visited. Users will not be able to select a new patient from within VistAWeb, but may return to CPRS to select a new patient. This new patient will then be viewable in VistAWeb automatically.
- Activity Logging—VistAWeb tracks the user's movements through the application in a SQL Server database table.
- Error Logging / Email—whenever an error occurs with the application, an email is automatically sent to the VistAWeb Technical Support Group.
- Multithreaded Site Connections—when a user identifies what patient data he/she wants to see, MDO creates a separate connection thread to each site where the patient has data and retrieves the data asynchronously rather than iteratively.
- No Caching—pages that present patient data are prevented from being cached in the client-side Temporary Internet Files folder, thereby preventing users from retrieving patient data after the VistAWeb session has terminated.
- Session Timeout—the VistAWeb browser session times out after 15 minutes of inactivity. A two-minute warning window will pop up, allowing the user the option to terminate the session early or continue working, the latter choice thereby resetting the timeout period.

Using VistAWeb User Management Web Application

The VistAWebUserMgt application is an intranet application that is used by select individuals to grant VistAWeb users the ability to select patients from remote sites. Note that access to this application is restricted to only a handful of individuals who have been identified to have the proper credentials necessary to assign remote site permissions to VistAWeb users.

Process

- Once a VistAWeb user has submitted a request to be able to select patients from one or more remote sites, the associated authorizer will use the application to grant the approved site permissions. The authorizer must provide a reason for each entry; the reasons are typically provided by the VistAWeb user to the local Information Security Officer (ISO) when the user makes the initial request.
- The data is saved to a SQL server database table.
- The associated authorizer notifies the user and the local ISO of the authorized permissions.

Like the VistAWeb EMR project, the VistAWeb User Management project is governed by the same development methodology.

SQL Server Database Overview

VistAWeb needs the following tables in order to run and will interact with these tables constantly. The EMR database contains the following tables:

- *CprsUsers*—CPRS-spawned VistAWeb checks this table to see if the CPRS user has logged into the spawned version before. If not, then the user is asked to log into VistAWeb. Once this is done one time, the user's information is added to the CPRSUsers table. Future CPRS-spawned VistAWeb browsers will not ask the user to log in if their information is found in this table.
- *SpecialUsers*—retains the remote site permissions assigned to users.
- *Log*—tracks the movements of users within the VistAWeb application.
- *Request*—tracks remote sites to which users want special user access.

Appendix A: Database Schema

- Database Name: EMR
- Database Tables:
 - Log
 - CprsUsers
 - SpecialUser
 - Requests
- Views:
 - LogDesc

Log Creation Script

```
CREATE TABLE [Log] (
    [id] [numeric](19, 0) IDENTITY (1, 1) NOT NULL ,
    [requestDate] [datetime] NULL ,
    [remoteAddr] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [userId] [numeric](19, 0) NULL ,
    [userName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [userSitecode] [varchar] (6) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [userSitename] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS
    NULL ,
    [requestPage] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS      NULL
    ,
    [requestSitecode] [varchar] (6) COLLATE SQL_Latin1_General_CP1_CI_AS      NULL
    ,
    [requestSitename] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS  NULL
    ,
    [patientID] [numeric](19, 0) NULL ,
    [patientName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS      NULL
    ,
    [patientSensitivity] [tinyint] NULL ,
    [message] [varchar] (1000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    CONSTRAINT [PK_Log] PRIMARY KEY CLUSTERED
    (
        [id]
    ) ON [PRIMARY]
) ON [PRIMARY]
GO
```

CprsUsers Creation Script

```
CREATE TABLE [CprsUsers] (
    [UserID] [numeric](19, 0) NOT NULL ,
    [Sitecode] [varchar] (3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
```

```
[DUZ] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,  
[SSN] [varchar] (9) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,  
[Name] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL  
) ON [PRIMARY]  
GO
```

SpecialUsers Creation Script

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[SpecialUsers]') and  
OBJECTPROPERTY(id, N'IsUserTable') = 1)  
drop table [dbo].[SpecialUsers]  
GO  
CREATE TABLE [dbo].[SpecialUsers] (  
    [RecID] [numeric](19, 0) IDENTITY (1, 1) NOT NULL ,  
    [UserSiteID] [varchar] (3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,  
    [DUZ] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,  
    [UserName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,  
    [Site] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,  
    [Reason] [varchar] (200) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,  
    [ActiveDate] [datetime] NULL ,  
    [DeactiveDate] [datetime] NULL  
) ON [PRIMARY]  
GO
```

Requests Creation Script

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[Requests]') and  
OBJECTPROPERTY(id, N'IsUserTable') = 1)  
drop table [dbo].[Requests]  
GO  
CREATE TABLE [dbo].[Requests] (  
    [requestID] [numeric](19, 0) IDENTITY (1, 1) NOT NULL ,  
    [userID] [varchar] (3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,  
    [text] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL  
) ON [PRIMARY]  
GO
```

LogDesc View Creation Script

```
CREATE VIEW dbo.LogDesc  
AS  
SELECT *  
FROM dbo.Log  
ORDER BY id DESC
```

Appendix B: VistAWeb Code Samples

Sample HTM (Countdown.htm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>Countdown</title>
        <link rel="stylesheet" type="text/css" href="resources/css/main.css"></link>
        <script LANGUAGE="JavaScript">

var now = new Date();
var event = new Date();
event.setMinutes(event.getMinutes() + 2);
var seconds = (event - now) / 1000;
var showSeconds;
ID=window.setTimeout("update()", 1000);

function update() {
    now = new Date();
    seconds = (event - now) / 1000;
    seconds = Math.round(seconds);
    if (seconds > 59) {
        showSeconds = seconds - 60;
    } else {
        showSeconds = seconds;
        document.Countdown.minutes.value = 0;
    }

    document.Countdown.seconds.value = showSeconds;
    if (seconds == 0)
        returnToCaller("Close");
    else
        ID=window.setTimeout("update()",1000);
}

function returnToCaller(option) {
    window.returnValue = option;
    window.close();
}
</script>
<style>
body
{
margin-left: 20px;
}
</style>
</HEAD>
<body MS_POSITIONING="GridLayout">
<table border="0" width="100%">
<tr>
<td></td>
<td align="right"><IMG src="resources/images/Timeout.gif" alt="Timeout image"></td>
</tr>
</table>
<hr color="#cc0000">
```

```

<p>
    <form name="Countdown" method="post">
        VistaWeb has not been used for 15 minutes. It will close in the indicated time
        unless you click the "Don't Close" button...
        <p>
            <span class="hdr">Time Remaining Until VistaWeb Closes:</span><br>
            <br>
        <TABLE BORDER="5" CELLSPACING="5" CELLPADDING="0">
            <TR>
                <TD ALIGN="middle" WIDTH="23%><B>Mins:</B></TD>
                <TD ALIGN="middle" WIDTH="23%><B>Secs:</B></TD>
            </TR>
            <TR>
                <TD ALIGN="middle"><INPUT type="Text" name="minutes" size="2" value="1"></TD>
                <TD ALIGN="middle"><INPUT type="Text" name="seconds" size="2"></TD>
            </TR>
        </TABLE>
        <br>
        <input type="button" class="myButton" onclick="returnToCaller('Continue')" value="Don't
close VistaWeb">
        <input type="button" class="myButton" onclick="returnToCaller('Close')" value="Close
VistaWeb">
    </form>
</body>
</HTML>

```

Sample ASPX Page (TextRecord.aspx):

```

<%@ Page language="c#" Codebehind="TextRecordPage.aspx.cs" AutoEventWireup="false"
Inherits="EMR.TextRecordPage" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <script language="javascript">
            function DeselectDates()
            {
                document.forms[0].txtFrom.value = "";
                document.forms[0].txtTo.value = "";
            }
        </script>
        <title>TextRecordPage</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
        <link href="resources/css/main.css" type="text/css" rel="stylesheet">
        <script language="javascript" src="resources/scripts/js/functions_lib.js"></script>
        <script language="javascript" src="resources/scripts/js/JSExtras.js"></script>
    </HEAD>
    <body MS_POSITIONING="GridLayout" onload="init('TextRpt',
");setBehavior();hideNavigationPopupWarning()">
        <form id="TextRecordPage" method="post" runat="server">
            <asp:Panel ID="DateRangePanel" Runat="server">
                <TABLE border="0">
                    <TR>

```

```

        <TD class="hdr">Date Range:</TD>
        <TD colSpan="8">
            <asp:radiobuttonlist id="DateRange"
Runat="server" RepeatDirection="Horizontal">
                <asp:ListItem
Value="1">Today</asp:ListItem>
                <asp:ListItem Value="8">One
Week</asp:ListItem>
                <asp:ListItem Value="15">Two
Weeks</asp:ListItem>
                <asp:ListItem Value="31">One
Month</asp:ListItem>
                <asp:ListItem Value="183">Six
Months</asp:ListItem>
                <asp:ListItem Value="366">
Selected="True">One Year</asp:ListItem>
                <asp:ListItem Value="732">Two
Years</asp:ListItem>
                <asp:ListItem Value="66666">All
Results</asp:ListItem>
            </asp:radiobuttonlist></TD>
        </TR>
        <TR>
            <TD></TD>
            <TD><SPAN class="hdr">From:</SPAN>&nbsp;&nbsp;
                <asp:textbox id="txtFrom" Runat="server"
Columns="12"></asp:textbox>&nbsp;&nbsp;
                <SPAN class="hdr">To:</SPAN>&nbsp;&nbsp;
                <asp:textbox id="txtTo" Runat="server"
Columns="12"></asp:textbox>&nbsp;
                (mm/dd/yyyy)&nbsp;&nbsp;
                <input type="button" onclick="DeselectDates()" value="Clear Dates" class="myButton">
            </TD>
            <TD align="left">
                <asp:button id="QueryButton" Runat="server"
Text="Query" CssClass="myButton"></asp:button></TD>
            </TR>
        </TABLE>
    </asp:Panel>
    <p>
        <asp:Literal ID="TextArea" Runat="server"></asp:Literal>
    </p>
    <br clear="all">
    <asp:TextBox ID="ImageFileName" Runat="server"
CssClass="hidelImageText"></asp:TextBox>
    <asp:TextBox ID="ImageFileAlt" Runat="server"
CssClass="hidelImageText"></asp:TextBox>
    </form>
    <form name="TimeoutForm" action="Timedout.aspx" target="_top">
    </form>
</body>
</HTML>

```

Sample Code-Behind Page (TextRecord.aspx.cs):

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Reflection;
using System.Security;
using gov.va.med.vistaweb.util;

namespace EMR
{
    /// <summary>
    /// Summary description for TextRecordPage.
    /// </summary>
    public class TextRecordPage : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.RadioButtonList DateRange;
        protected System.Web.UI.WebControls.TextBox txtFrom;
        protected System.Web.UI.WebControls.TextBox txtTo;
        protected System.Web.UI.WebControls.Button QueryButton;
        protected System.Web.UI.WebControls.Literal TextArea;
        protected System.Web.UI.WebControls.TextBox ImageFileName;
        protected System.Web.UI.WebControls.TextBox ImageFileAlt;
        protected System.Web.UI.WebControls.Panel DateRangePanel;

        private String sessionPageName = "";
        private String sessionDateRangeName = "";
        private String errMsg = "";
        private bool fDateRange = true;

        private void Page_Load(object sender, System.EventArgs e)
        {
            Response.CacheControl = "no-cache";
            Response.AddHeader("Pragma", "no-cache");
            Response.Expires = -1;

            if (Session["Patient"] == null)
            {
                Response.Redirect("ChangeFrameset.htm?Set=NoSession.htm",true);
            }

            String urlArgs = Request.QueryString.ToString();
            String[] args = StringUtils.Split(urlArgs(StringUtils.AMPERSAND));
            String[] flds = StringUtils.Split(args[0],StringUtils.EQUALS);
            sessionPageName = flds[1];
            sessionDateRangeName = sessionPageName + "DateRangeState";
            ImageFileName.Text = sessionPageName;
            errMsg = "TextRecordPage.aspx: " + sessionPageName + "\n\n";
            flds = StringUtils.Split(args[1],StringUtils.EQUALS);
            flds[1] = flds[1].Replace("+", " ");
        }
    }
}
```

```

        ImageFileAlt.Text = flds[1];
        flds = StringUtils.Split(args[2],StringUtils.EQUALS);
        fDateRange = flds[1].Equals("Y");
        if (!fDateRange)
        {
            DateRangePanel.Visible = false;
            //                                         DateRange.Visible = false;
            //                                         txtFrom.Visible = false;
            //                                         txtTo.Visible = false;
            //                                         QueryButton.Visible = false;
        }
        if (txtFrom.Text != "") DateRange.SelectedIndex = -1;
        if (!Page.IsPostBack)
        {
            query();
        }
    }
    private void query()
    {
        MDO.TimeInterval t = null;
        if (fDateRange)
        {
            t = Global.getTimeInterval(txtFrom.Text,txtTo.Text,DateRange);
            if (t == null)
            {
                Response.Write("Invalid from date: Must have format mm/dd/yyyy");
                Response.End();
            }
        }
        MDO.TextRecord[] rex = null;
        TextArea.Text = "";
        try
        {
            rex = getRex(t);
            string s = "";
            if (rex.Length != 0)
            {
                s += "<ul id=\"siteReportsMenu\">";
                for (int z=0; z<rex.Length; z++)
                {
                    s += "<li><a href=\"#" + id +"\" id=\"site" + z + "\"";
                    class="menuLinks">" + rex[z].getSite().getName() + "</a></li>";
                }
                s += "</ul>";
            }
            for (int i=0; i<rex.Length; i++)
            {
                s += "<div id=\"siteDiv" + i + "\" class=\"sitePanel\">";
                s += "<h3>" + rex[i].getSite().getName() + "</h3>";
                s += "<pre>" + rex[i].getText() + "</pre>";
                s += "</div>";
            }
            TextArea.Text = s;
        }
        catch (Exception ex)
        {

```

```

        Global.showOops((MDO.User)Session["User"],(MDO.Patient)Session["Patient"],this.Context.Request,ex,
                        this.Context.Response,errMsg);
                }
                Log.write(Request,Session,null,"From " +
                ((MDO.MultiSiteDAO)Session["MultiSiteDao"]).getSiteList());
            }
            private MDO.TextRecord[] getRex(MDO.TimeInterval interval)
            {
                MDO.Patient patient = (MDO.Patient)Session["Patient"];
                MDO.TextRecord[] rex = null;
                try
                {
                    Type patientClass = patient.GetType();
                    String methodName = "get" + sessionPageName;
                    MethodInfo patientGetMethod = patientClass.GetMethod(methodName);
                    rex = (MDO.TextRecord[])patientGetMethod.Invoke(patient,null);
                    if (rex == null)
                    {
                        MDO.MultiSiteDAO multiSiteDao =
                        (MDO.MultiSiteDAO)Session["MultiSiteDAO"];
                        Type multiSiteDaoClass = multiSiteDao.GetType();
                        Type[] argTypes = null;
                        Object[] args = null;
                        if (fDateRange)
                        {
                            argTypes = new Type[2];
                            argTypes[0] = patientClass;
                            argTypes[1] = interval.GetType();
                            args = new Object[2];
                            args[0] = patient;
                            args[1] = interval;
                        }
                        else
                        {
                            argTypes = new Type[1];
                            argTypes[0] = patientClass;
                            args = new Object[1];
                            args[0] = patient;
                        }
                        MethodInfo multiSiteDaoMethod =
                        multiSiteDaoClass.GetMethod("set" + sessionPageName,argTypes);
                        multiSiteDaoMethod.Invoke(multiSiteDao,args);
                        rex = (MDO.TextRecord[])patientGetMethod.Invoke(patient,null);
                        Session[sessionDateRangeName] = new
                        DateRangeState(DateRange.SelectedIndex,txtFrom.Text,txtTo.Text);
                    }
                    else
                    {
                        DateRangeState drs =
                        (DateRangeState)Session[sessionDateRangeName];
                        DateRange.SelectedIndex = drs.dateRangeIndex;
                        txtFrom.Text = drs.fromDate;
                        txtTo.Text = drs.toDate;
                    }
                }

```

```

        }
        catch (Exception ex)
        {
            Global.showOops((MDO.User)Session["User"],patient,this.Context.Request,ex,this.Context.Response,errMsg);
        }
        return rex;
    }
    private void QueryButton_Click(object sender, System.EventArgs e)
    {
        MDO.Patient patient = (MDO.Patient)Session["Patient"];
        MDO.TextRecord[] empty = new MDO.TextRecord[0];
        try
        {
            Type patientClass = patient.GetType();
            String methodName = "set" + sessionPageName;
            Type[] argTypes = {empty.GetType()};
            MethodInfo theMethod =
                patientClass.GetMethod(methodName,argTypes);
            empty = null;
            Object[] args = {empty};
            theMethod.Invoke(patient,args);
        }
        catch (Exception ex)
        {
            Global.showOops((MDO.User)Session["User"],patient,this.Context.Request,ex,this.Context.Response,errMsg);
        }
        query();
    }
    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
        //
        // CODEGEN: This call is required by the ASP.NET Web Form Designer.
        //
        InitializeComponent();
        base.OnInit(e);
    }

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.QueryButton.Click += new System.EventHandler(this.QueryButton_Click);
        this.Load += new System.EventHandler(this.Page_Load);
    }
    #endregion
}

```

Sample Template Page (TextRecord.aspx.resx)

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
    <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
        <xsd:element name="root" msdata:IsDataSet="true">
            <xsd:complexType>
                <xsd:choice maxOccurs="unbounded">
                    <xsd:element name="data">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="value"
                                    type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
                                <xsd:element name="comment"
                                    type="xsd:string" minOccurs="0" msdata:Ordinal="2" />
                                    </xsd:sequence>
                                    <xsd:attribute name="name" type="xsd:string" />
                                    <xsd:attribute name="type" type="xsd:string" />
                                    <xsd:attribute name="mimetype"
                                        type="xsd:string" />
                                </xsd:complexType>
                            </xsd:element>
                            <xsd:element name="resheader">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="value"
                                            type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
                                            </xsd:sequence>
                                            <xsd:attribute name="name" type="xsd:string"
                                                use="required" />
                                                </xsd:complexType>
                                            </xsd:element>
                                        </xsd:choice>
                                    </xsd:complexType>
                                </xsd:element>
                            </xsd:schema>
                            <resheader name="ResMimeType">
                                <value>text/microsoft-resx</value>
                            </resheader>
                            <resheader name="Version">
                                <value>1.0.0.0</value>
                            </resheader>
                            <resheader name="Reader">
                                <value>System.Resources.ResXResourceReader, System.Windows.Forms,
                                    Version=1.0.3102.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
                            </resheader>
                            <resheader name="Writer">
                                <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
                                    Version=1.0.3102.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
                            </resheader>
                        </root>
```